# CPU

## Corso di Calcolatori Elettronici A

## 2007/2008

**Sito Web:http://prometeo.ing.unibs.it/quarella**

***Prof. G. Quarella***

**prof@quarella.net**

# Introduzione

Nella progettazione dell'unità di elaborazione (datapath) e dell'unità di controllo che implementa il set di istruzioni MIPS, si considera un sottoinsieme delle istruzioni composto da:

**lw** e **sw**

**add, sub, and, or** e **slt**

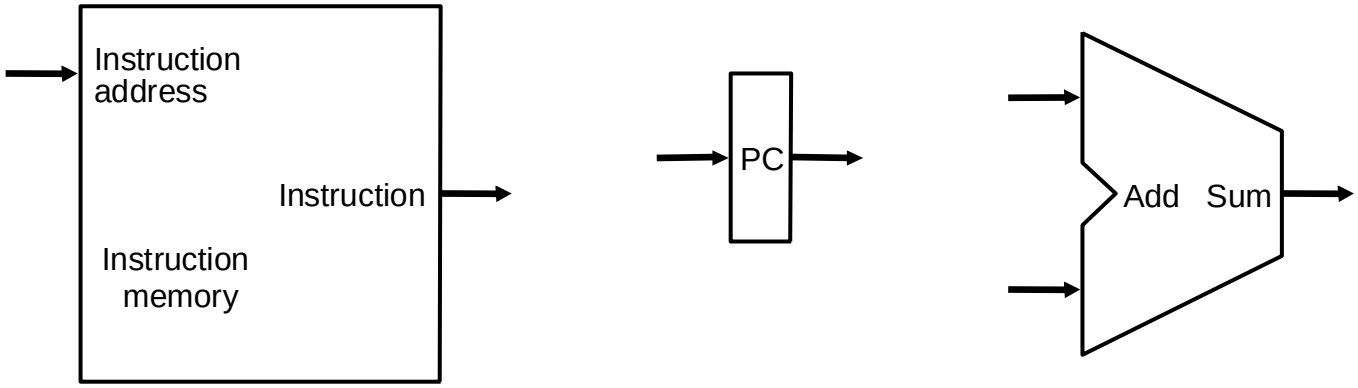
**beq** e **j**

Si considerano i seguenti principi progettuali:

1. Semplicità e regolarità
2. Minori sono le dimensioni, maggiore è la velocità
3. Un buon progetto richiede buoni compromessi
4. Rendere veloce l'evento più frequente

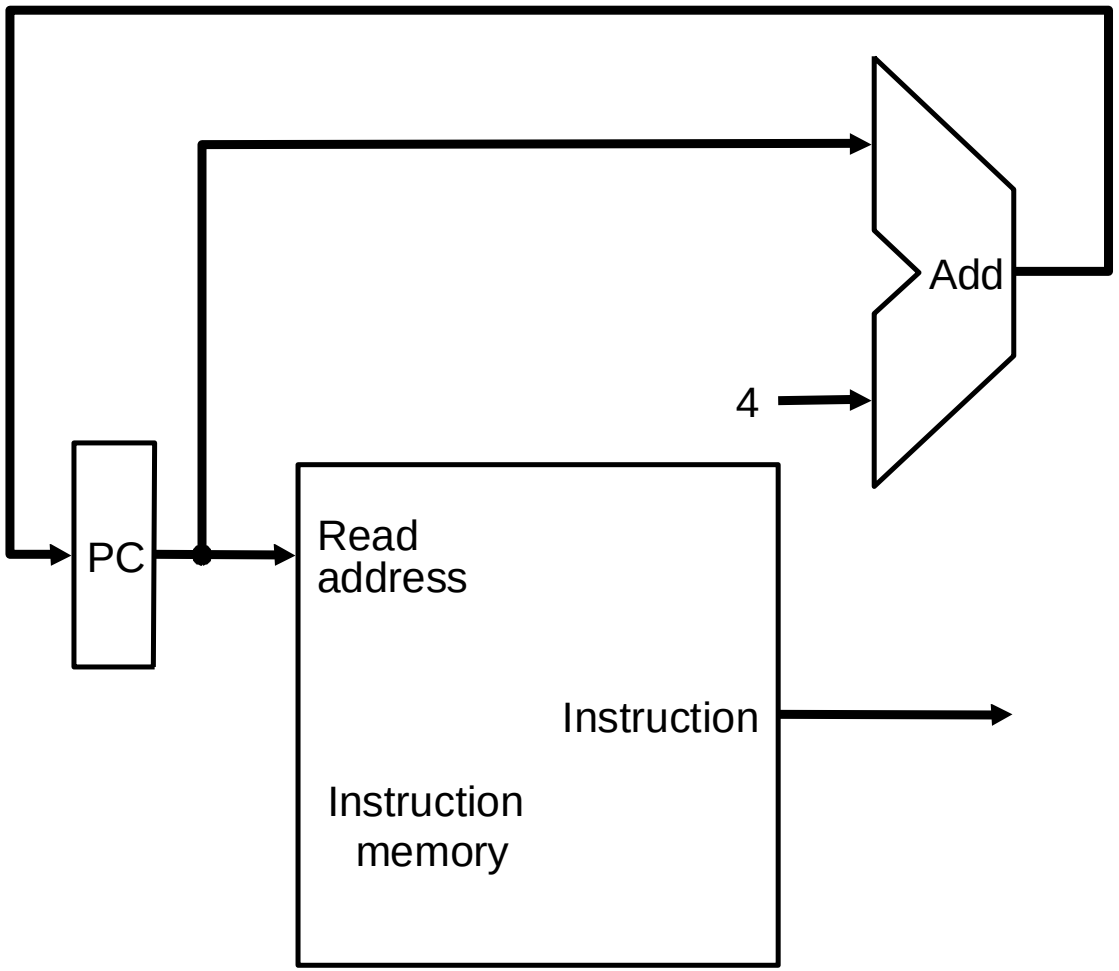
# CPU a singolo ciclo

Elementi necessari per progettare l'unità di elaborazione che implementa il set MIPS ridotto.

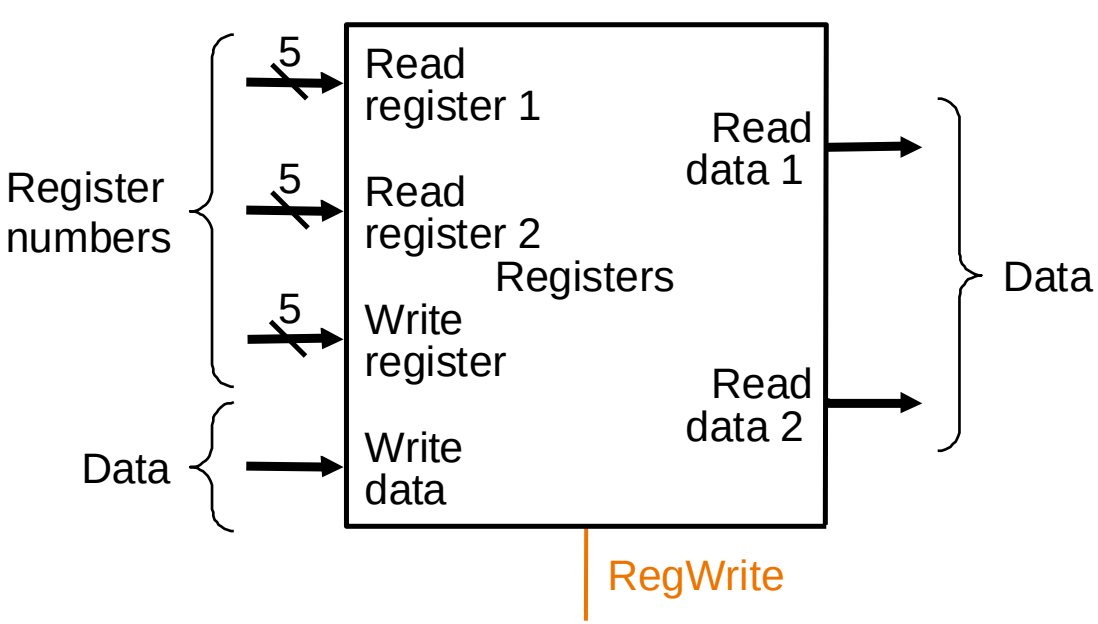



a. Instruction memory b. Program counter c. Adder

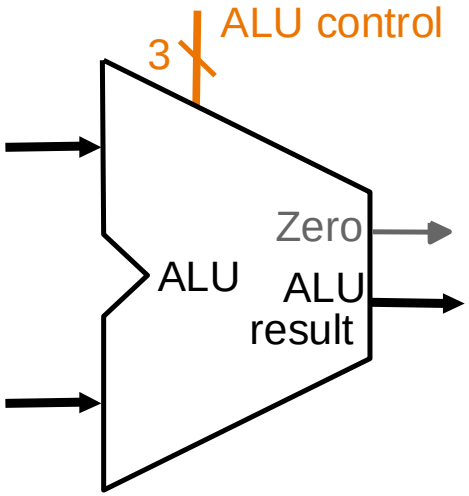
# Fetch



# Formato R

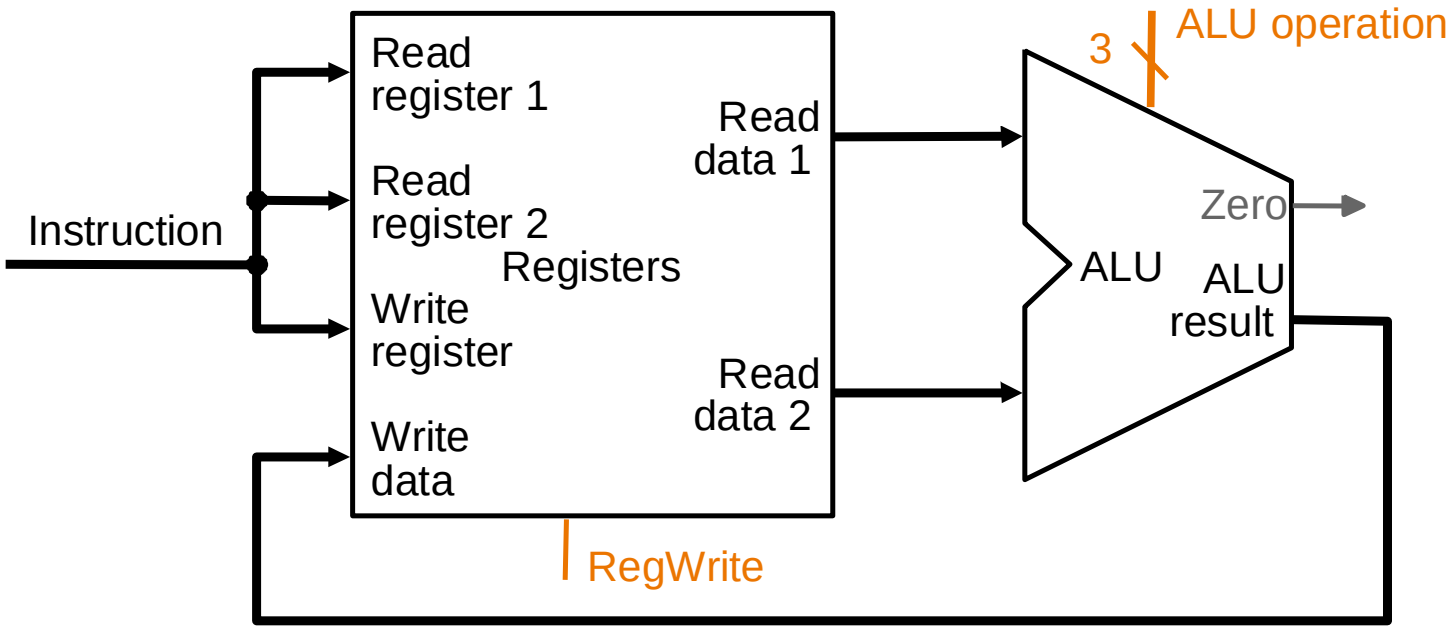



a. Registers

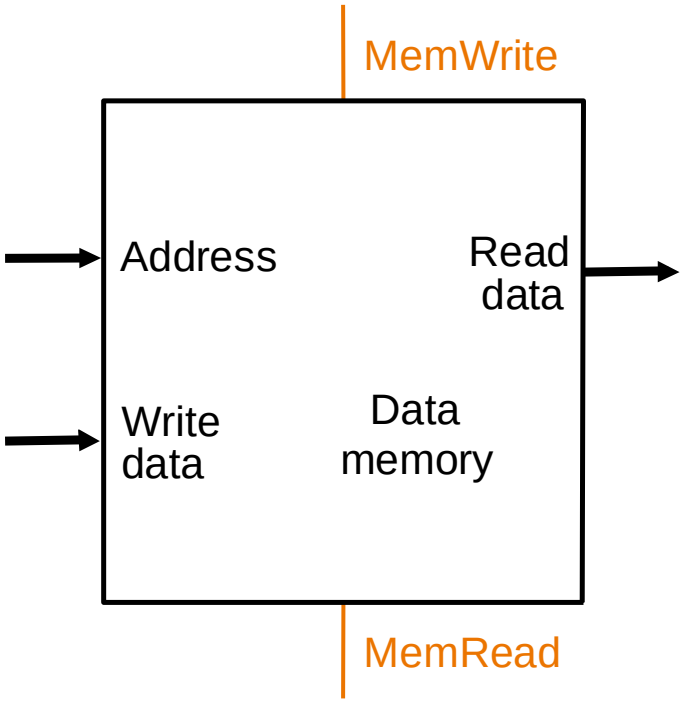



b. ALU

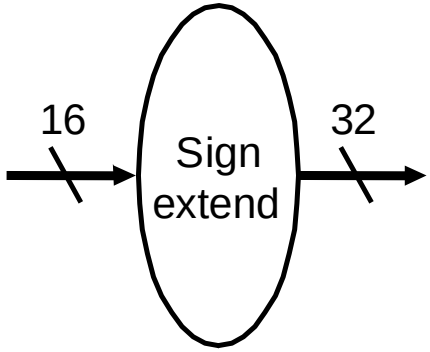
# Formato R


Read
register 1
Read
register 2
Registers
Write
register
Write
data
Read
data 1
Read
data 2
RegWrite
Instruction
3
ALU operation
ALU
Zero
ALU
result

# Load - store

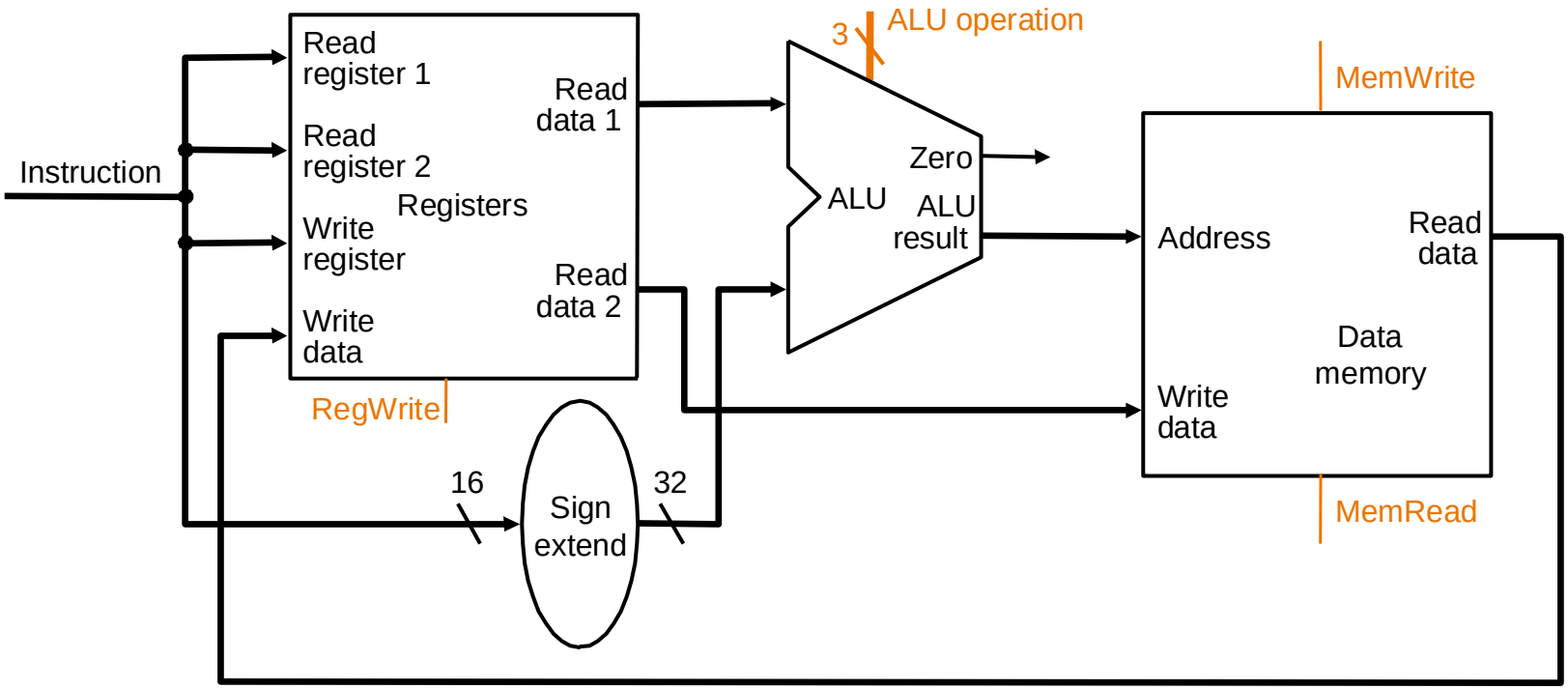



a. Data memory unit

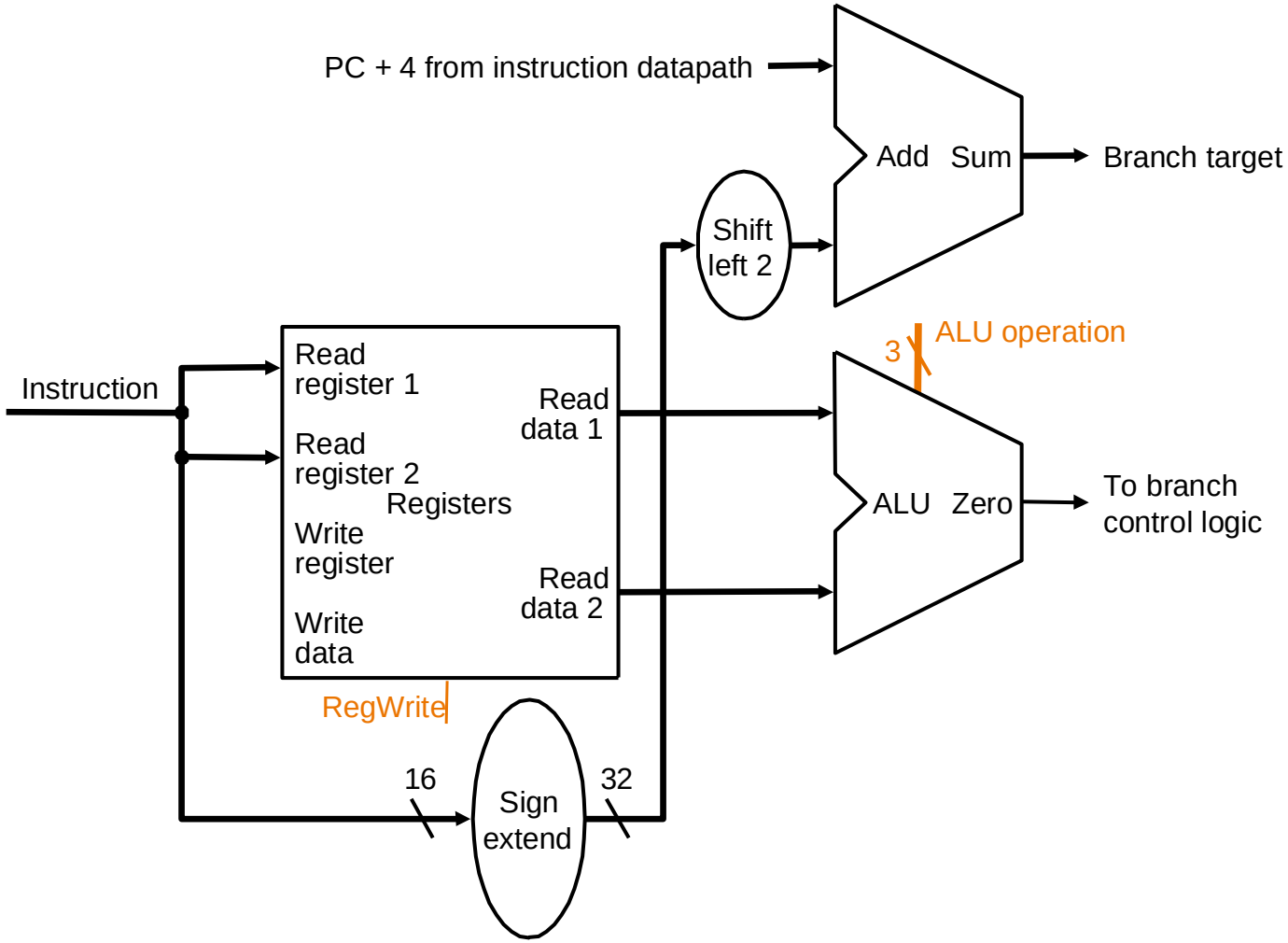



b. Sign-extension unit

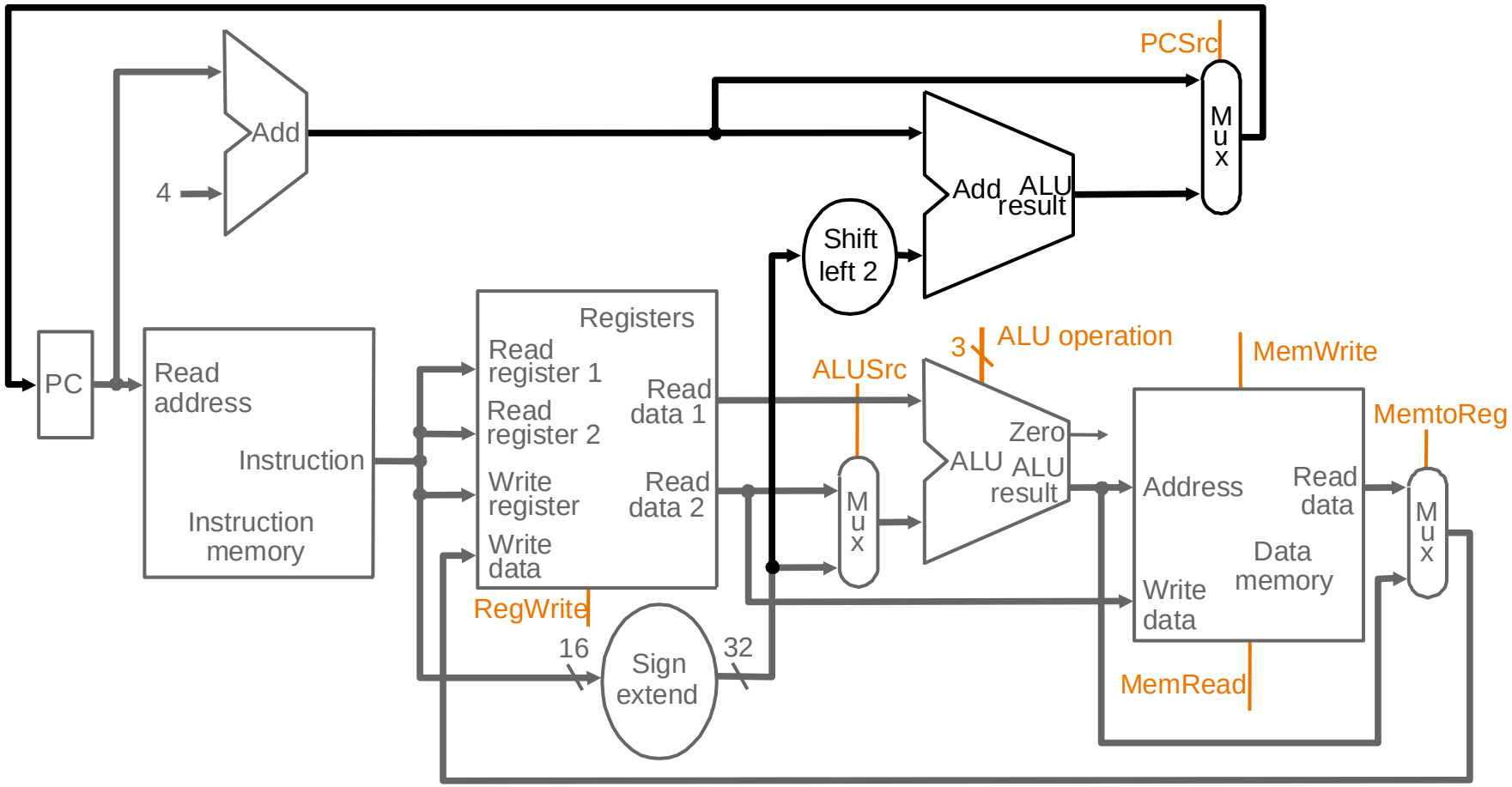
# Load - store


Instruction
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
RegWrite
3
ALU operation
ALU
Zero
ALU result
16
Sign extend
32
MemWrite
Address
Read data
Data memory
Write data
MemRead

# Salti condizionati



# Unità di elaborazione elementare



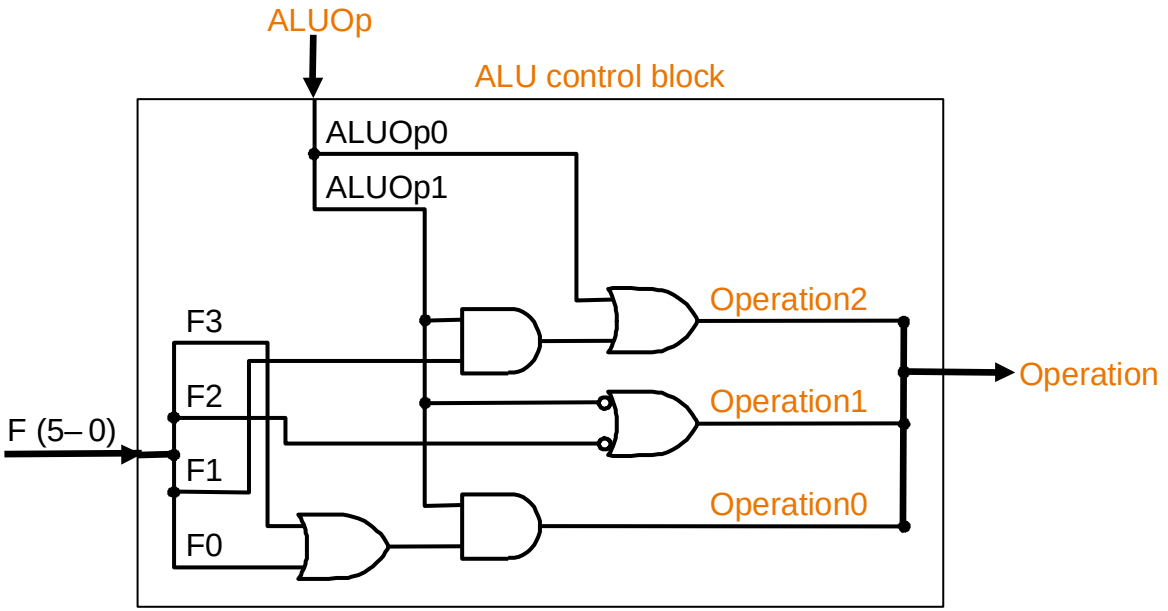
# Unità di controllo per la ALU

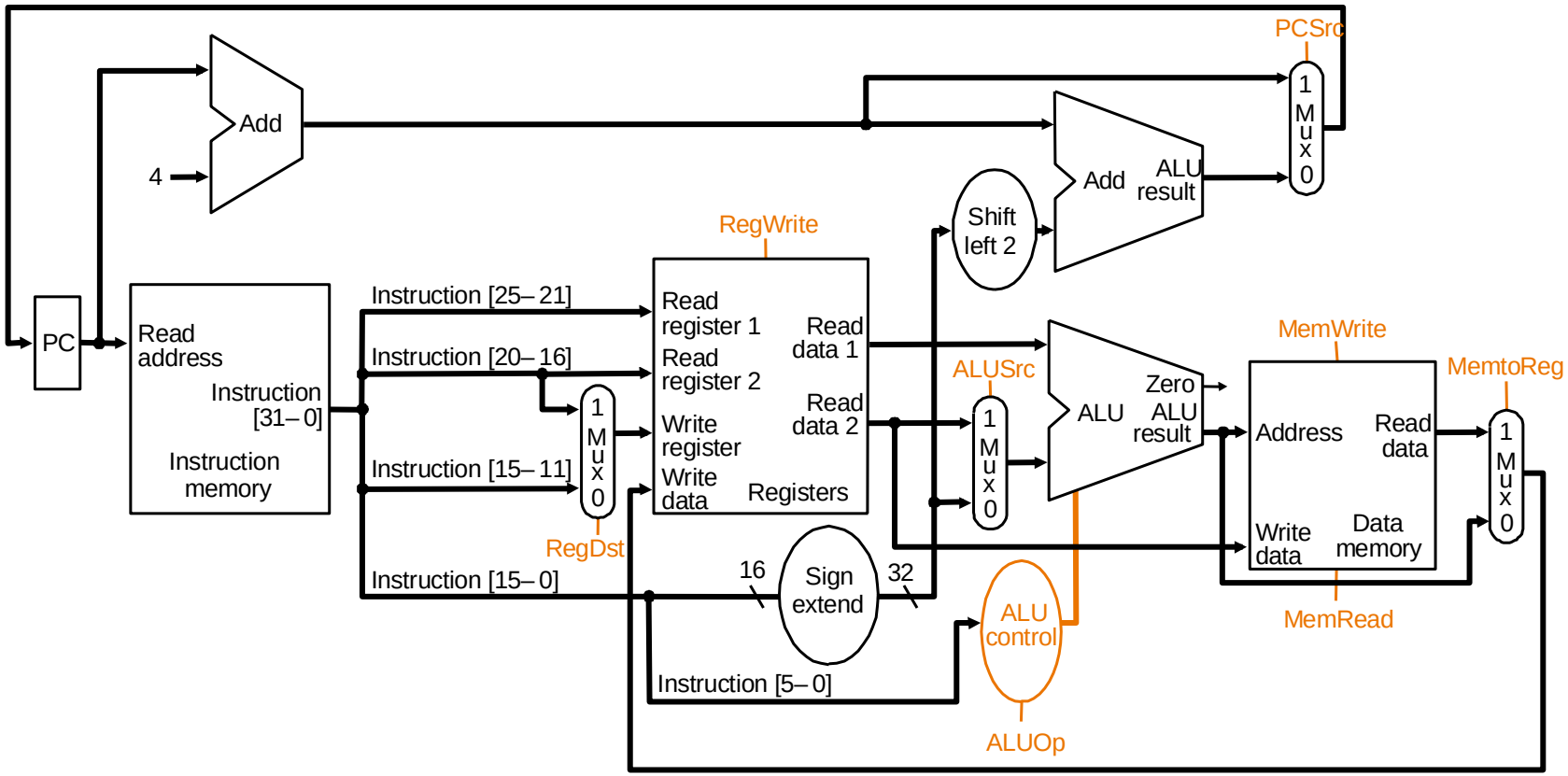
| Ingresso di controllo della ALU | Funzione |
|---|---|
| 000 | AND |
| 001 | OR |
| 010 | add |
| 110 | subtract |
| 111 | set-on-less-than |

In ingresso si ha il campo funzione dell'istruzione e un ingresso ALUOp di 2 bit che specifica quale operazione eseguire in base al tipo di istruzione (00-somma, 01-sottrazione, 10-funct).

| Instruction opcode | ALUOp | | Funct field | | | | | | Operation |
|---|---|---|---|---|---|---|---|---|---|
| | ALUOp1 | ALUOp0 | F5 | F4 | F3 | F2 | F1 | F0 | |
| lw, sw | 0 | 0 | X | X | X | X | X | X | 010 |
| beq | X | 1 | X | X | X | X | X | X | 110 |
| add | 1 | X | X | X | 0 | 0 | 0 | 0 | 010 |
| sub | 1 | X | X | X | 0 | 0 | 1 | 0 | 110 |
| and | 1 | X | X | X | 0 | 1 | 0 | 0 | 000 |
| or | 1 | X | X | X | 0 | 1 | 0 | 1 | 001 |
| slt | 1 | X | X | X | 1 | 0 | 1 | 0 | 111 |

# Unità di controllo per la ALU




PCSrc
Add
4
1
M
u
x
0
Add
ALU
result
Shift
left 2
RegWrite
PC
Read
address
Instruction
[31– 0]
Instruction
memory
Instruction [25– 21]
Instruction [20– 16]
Instruction [15– 11]
Instruction [15– 0]
Read
register 1
Read
register 2
Write
register
Write
data
Read
data 1
Read
data 2
Registers
RegDst
ALUSrc
ALU
Zero
ALU
result
MemWrite
MemtoReg
Address
Read
data
Write
data
Data
memory
MemRead
16
Sign
extend
32
ALU
control
Instruction [5– 0]
ALUOp

# Formato istruzioni

**Formato-R** (istruzioni aritmetiche)

| op | rs | rt | rd | shamt | funct |
|---|---|---|---|---|---|
| 31-26 | 25-21 | 20-16 | 15-11 | 10-6 | 5-0 |

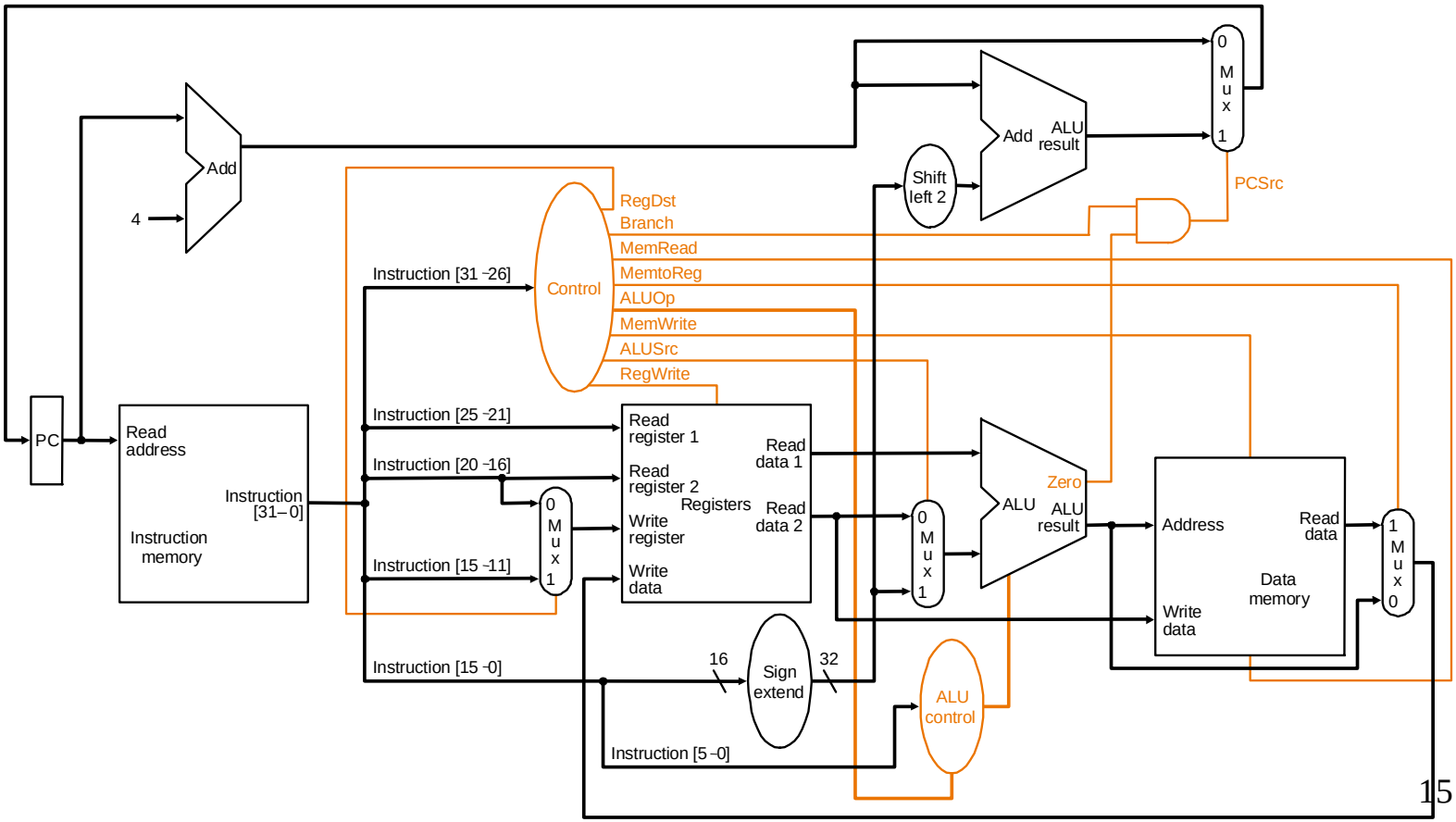
**Formato-I** (load, store, salto condizionato)

| op | rs | rt | address |
|---|---|---|---|
| 31-26 | 25-21 | 20-16 | 15-0 |

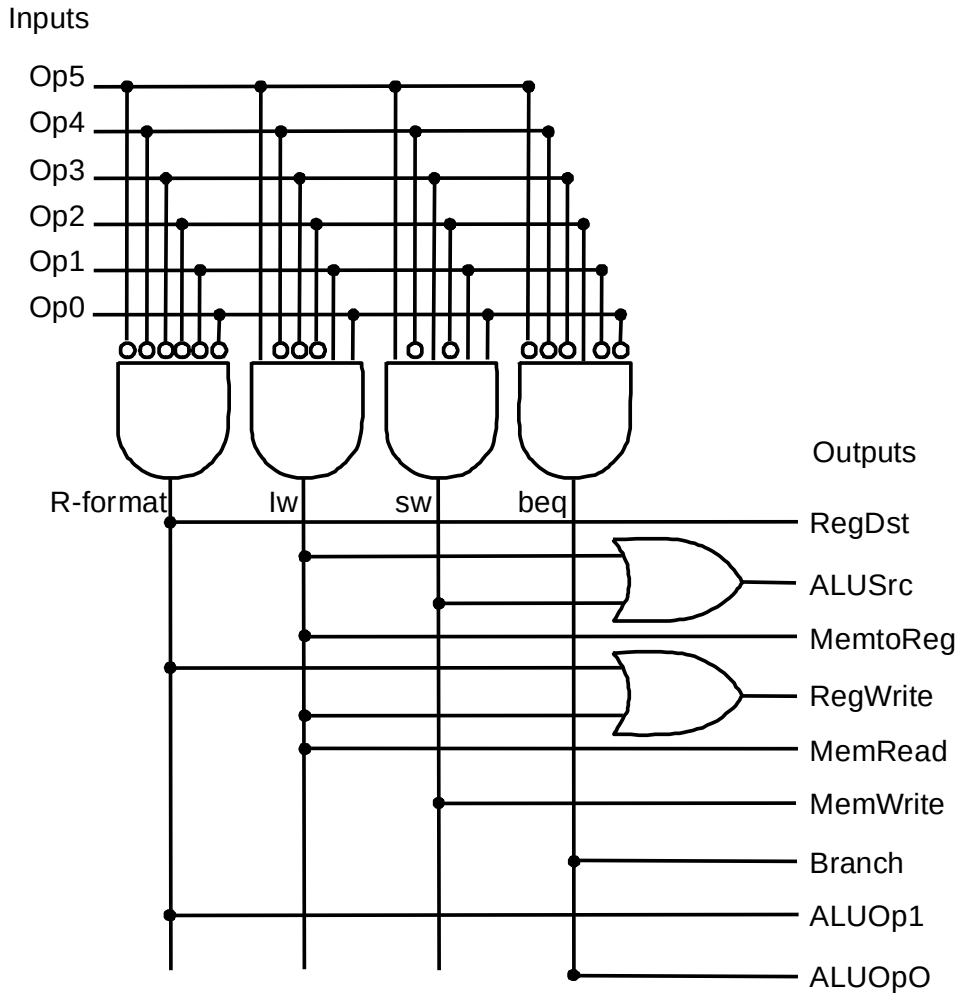
*rs*: registro base al cui contenuto va sommato address
*rt*: primo registro che compare nell'istruzione (registro destinazione per lw e registro sorgente per sw)

# Progetto dell'unità di controllo principale

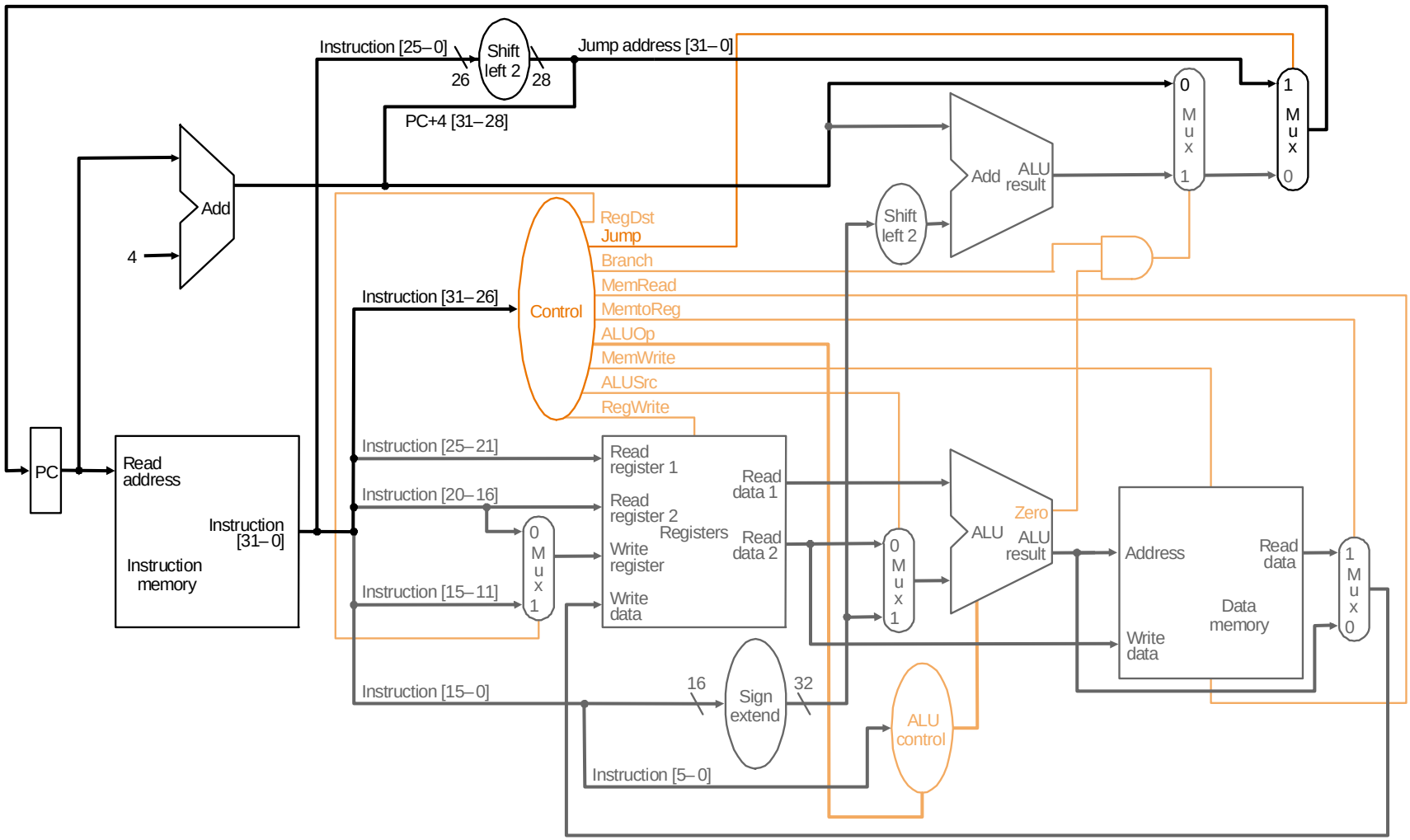
| Instruction | RegDst | ALUSrc | Memto-Reg | Reg Write | Mem Read | Mem Write | Branch | ALUOp1 | ALUp0 |
|---|---|---|---|---|---|---|---|---|---|
| R-format | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| `lw` | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| `sw` | X | 1 | X | 0 | 0 | 1 | 0 | 0 | 0 |
| `beq` | X | 0 | X | 0 | 0 | 0 | 1 | 0 | 1 |



# PLA della funzione di controllo principale per l’implementazione a singolo ciclo di clock



# Implementazione dell'istruzione jump



# Svantaggi dell'implementazione a singolo ciclo

In pratica l'implementazione a singolo ciclo non si utilizza per la sua inefficienza: infatti…

Tutte le istruzioni devono essere eseguite in un periodo di clock (CPI=1).

Il periodo di clock è determinato quindi dall'istruzione più lenta, ovvero da quella che utilizza più unità funzionali come la **load** (memoria istruzioni, register file, ALU, memoria dati e register file).

In definitiva non si può rendere veloce l'evento più frequente.

L'inefficienza non riguarda solo le prestazioni, ma anche il costo dell'hardware.

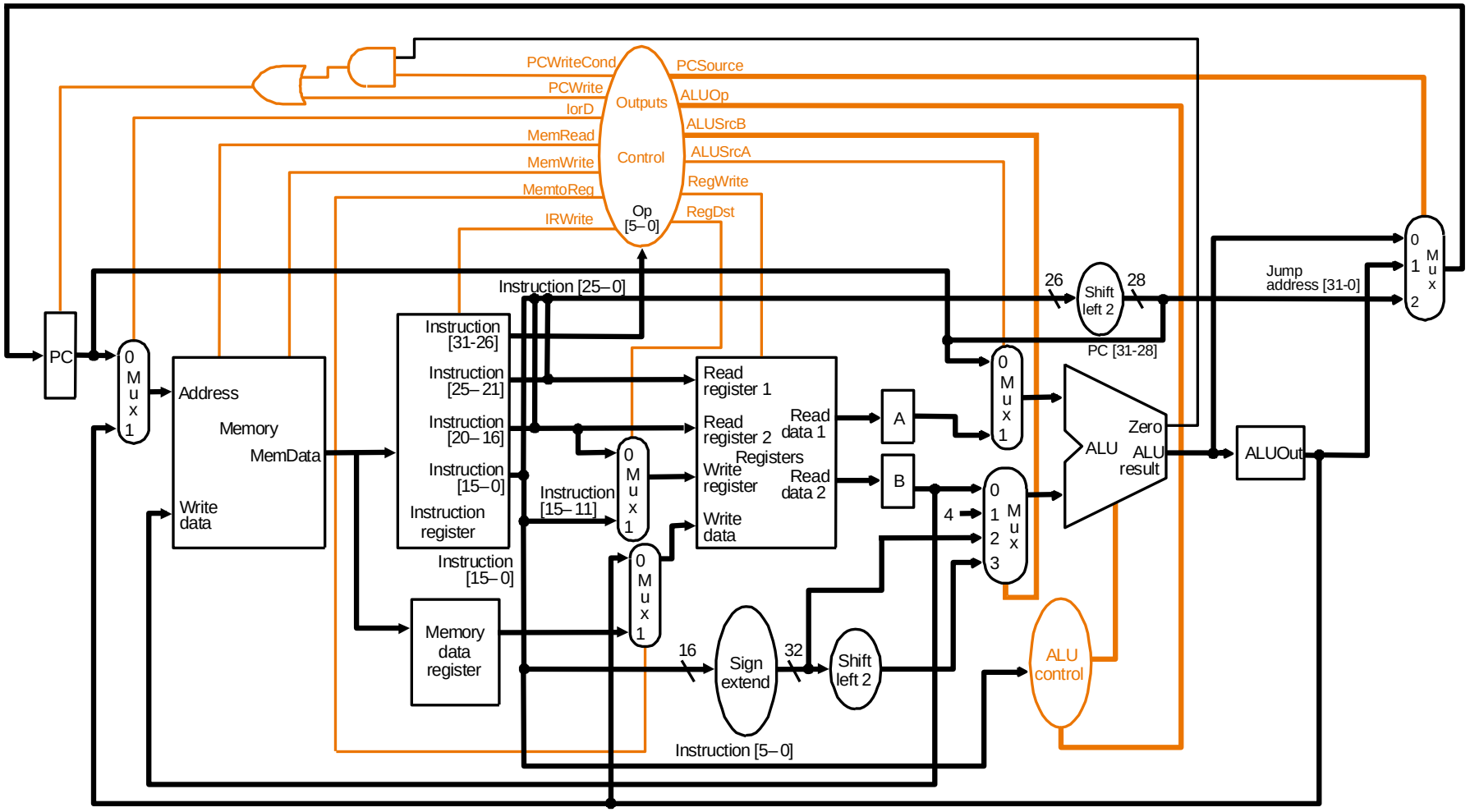
# Implementazione multi-ciclo

L’esecuzione di ogni istruzione è decomposta in una serie di passi ed ogni passo richiede un ciclo di clock.

La stessa unità funzionale si può riutilizzare in cicli di clock differenti. Si ha quindi una sola memoria per istruzioni e dati ed una sola ALU.

Bisogna introdurre dei registri aggiuntivi per memorizzare quei dati che devono essere disponibili in cicli di clock differenti (IR, MDR, A, B e ALUOut).

# Unità di elaborazione-datapath multi-ciclo



# Passi di esecuzione

L’esecuzione di un’istruzione può essere suddivisa in 5 passi:

1. **Prelievo dell’istruzione (Fetch)**
2. **Decodifica dell’istruzione e caricamento dei registri**
3. **Esecuzione, calcolo dell’indirizzo di memoria o completamento del salto**
4. **Accesso alla memoria o completamento dell’istruzione di tipo R**
5. **Completamento lettura da memoria**

L’esecuzione di un’istruzione può impiegare da 3 a 5 cicli di clock.

Per descrivere le operazioni svolte in ogni passo si utilizza la notazione **RTL** (**Register-Transfer Language**).

# 1. Prelievo dell’istruzione (Fetch)

```
IR = Memoria[PC];
PC = PC + 4;
```

# 2. Decodifica dell'istruzione e caricamento dei registri

```
A = Reg[IR[25-21]];
B = Reg[IR[20-16]];
ALUOut = PC + (sign-extend(IR[15-0]) << 2);
```

*Esecuzione di operazioni ottimistiche, grazie alla regolarità del formato delle istruzioni.*

Il tipo di istruzione sta per essere decodificato dalla logica di controllo.

# 3. Esecuzione, calcolo dell'indirizzo di memoria o completamento del salto

**La ALU sta eseguendo una delle 3 seguenti funzioni in base al tipo di istruzione:**

1. **Accesso alla memoria:**

```
ALUOut = A + sign-extend(IR[15-0]);
```

2. **Istruzione logico-aritmetica (tipo R):**

```
ALUOut = A op B;
```

3. **Salto condizionato (Branch):**

```
if (A==B) PC = ALUOut;
```

**Salto incondizionato (jump):**

```
PC = PC[31-28] || (IR[25-0] << 2);
```

# 4. Accesso alla memoria o completamento dell'istruzione di tipo R

**Load o Store**

```
MDR = Memoria[ALUOut];
      o
Memory[ALUOut] = B;
```

**Istruzione logico-aritmetica (tipo R):**

```
Reg[IR[15-11]] = ALUOut;
```

# 5. Completamento lettura da memoria

**Load:**

```
Reg[IR[20-16]]= MDR;
```

| Step name | Action for R-type instructions | Action for memory-reference instructions | Action for branches | Action for jumps |
|---|---|---|---|---|
| Instruction fetch | IR = Memory[PC]<br>PC = PC + 4 | | | |
| Instruction decode/register fetch | A = Reg [IR[25-21]]<br>B = Reg [IR[20-16]]<br>ALUOut = PC + (sign-extend (IR[15-0]) << 2) | | | |
| Execution, address computation, branch/ jump completion | ALUOut = A op B | ALUOut = A + sign-extend (IR[15-0]) | if (A ==B) then PC = ALUOut | PC = PC [31-28] II (IR[25-0]<<2) |
| Memory access or R-type completion | Reg [IR[15-11]] = ALUOut | Load: MDR = Memory[ALUOut]<br>or<br>Store: Memory [ALUOut] = B | | |
| Memory read completion | | Load: Reg[IR[20-16]] = MDR | | |

# Progetto dell'unità di controllo

Mentre nel caso di un'unità di elaborazione a singolo ciclo, l'unità di controllo non era altro che una semplice rete combinatoria, con un'implementazione **multi-ciclo** l'unità di controllo è più complessa, in quanto deve specificare sia segnali da affermare in ciascun passo sia il passo successivo in base al tipo dell'istruzione.

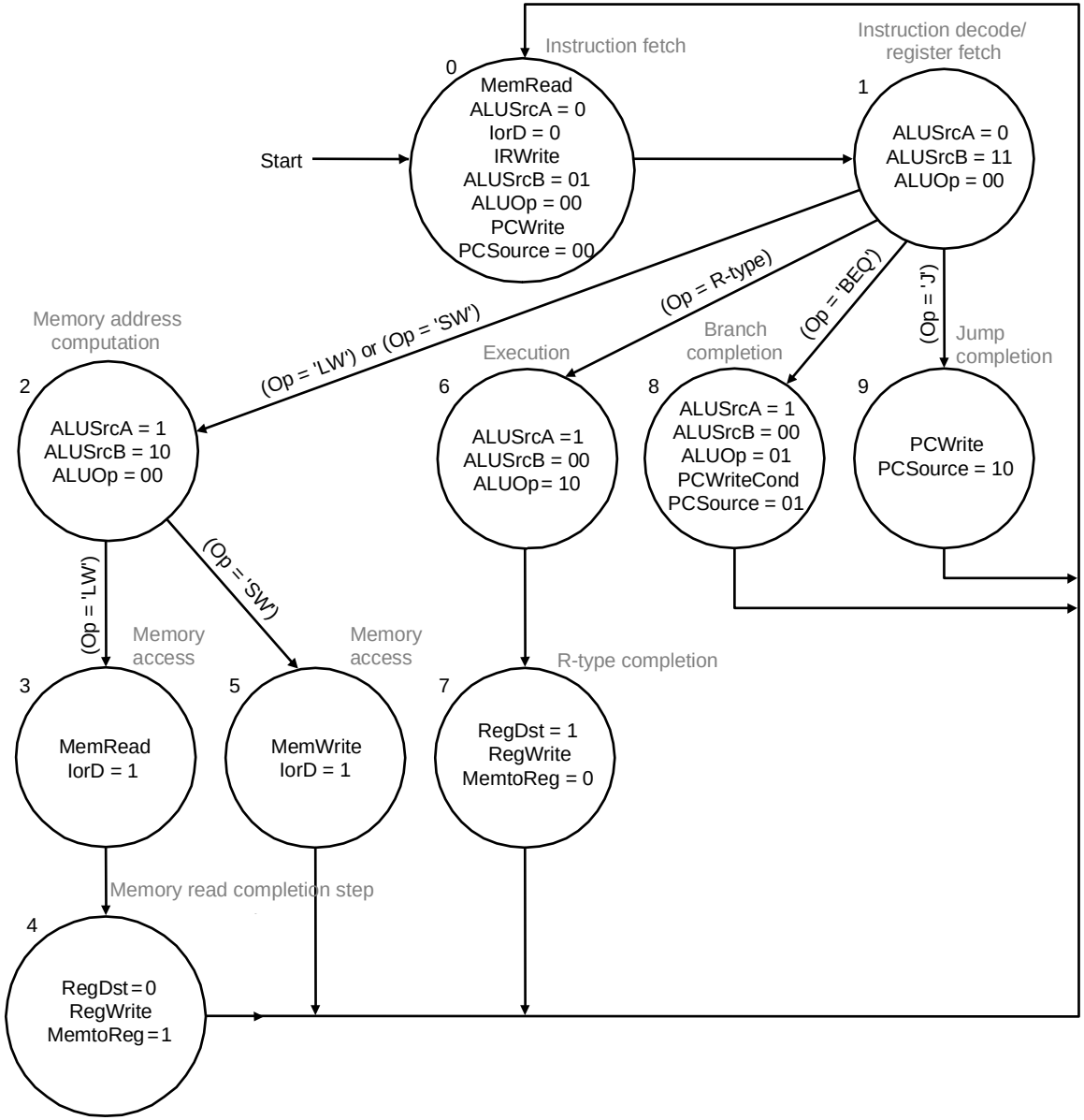
Si hanno due tecniche di specifica dell'unità di controllo:

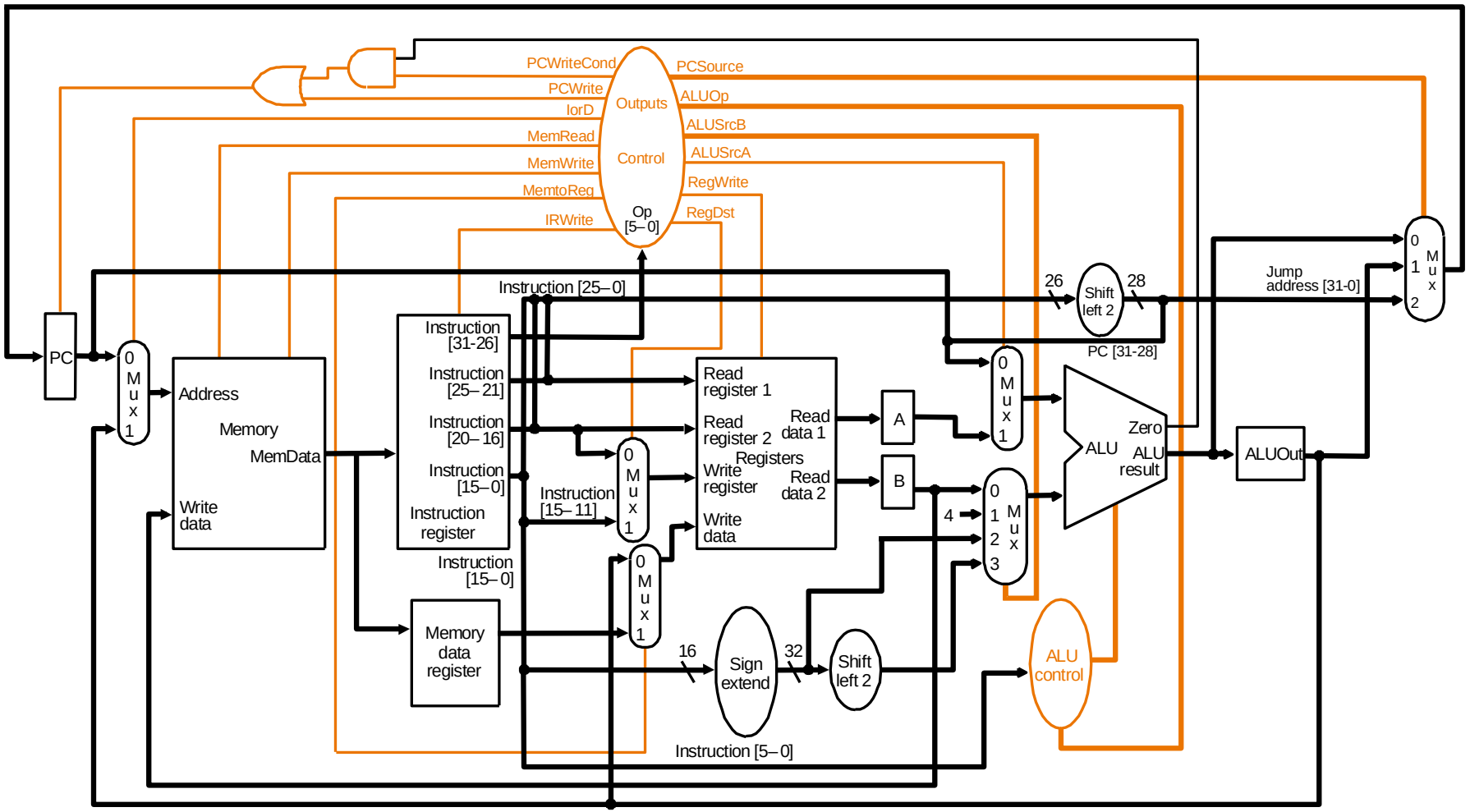
**1. Macchina a stati finiti** (FSM – Finite State Machine)

**2. Microprogrammazione**

Da tali specifiche si ottiene l'implementazione tramite porte logiche, ROM o PLA.

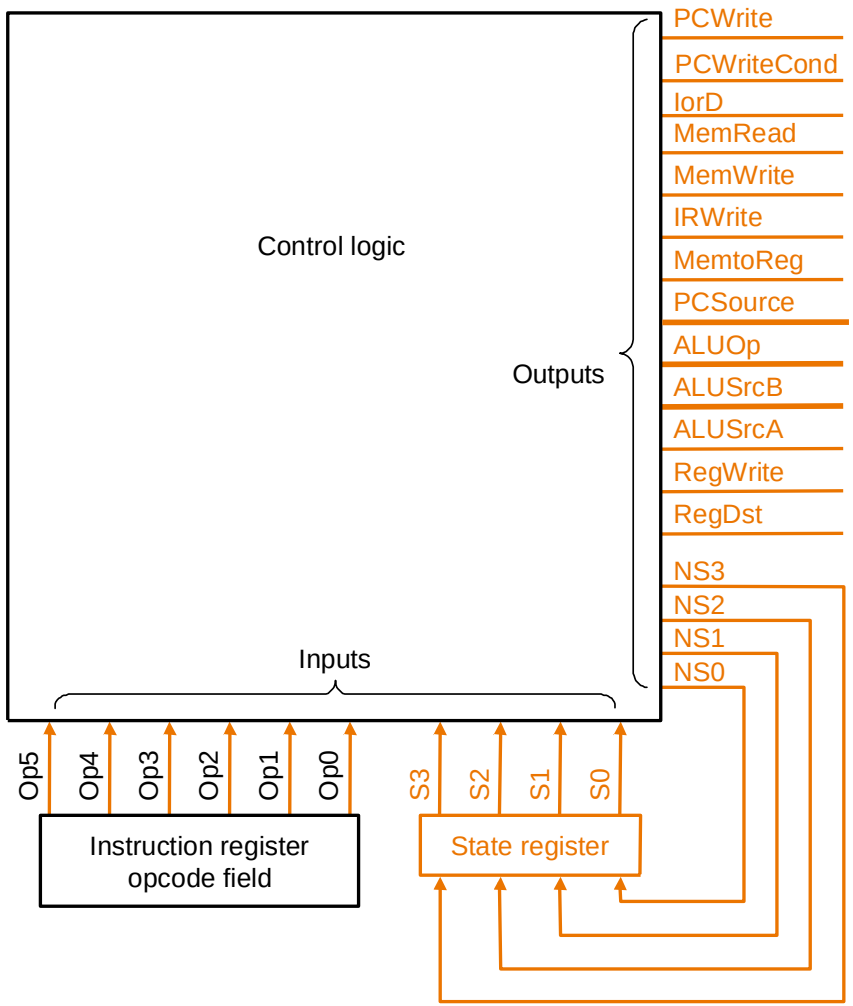
# Progetto dell'unità di controllo
# Macchina a stati finiti




PCWriteCond
PCWrite
IorD
MemRead
MemWrite
MemtoReg
IRWrite
Outputs
Control
Op
[5–0]
PCSource
ALUOp
ALUSrcB
ALUSrcA
RegWrite
RegDst
PC
0
M
u
x
1
Address
Memory
MemData
Write
data
Instruction
[31-26]
Instruction
[25–21]
Instruction
[20–16]
Instruction
[15–0]
Instruction
register
Instruction
[15–0]
Instruction [25–0]
Instruction
[15–11]
Memory
data
register
Read
register 1
Read
register 2
Registers
Write
register
Write
data
Read
data 1
Read
data 2
A
B
4
16
Sign
extend
32
Shift
left 2
26
Shift
left 2
28
PC [31-28]
ALU
Zero
ALU
result
ALUOut
ALU
control
Jump
address [31-0]
Instruction [5–0]

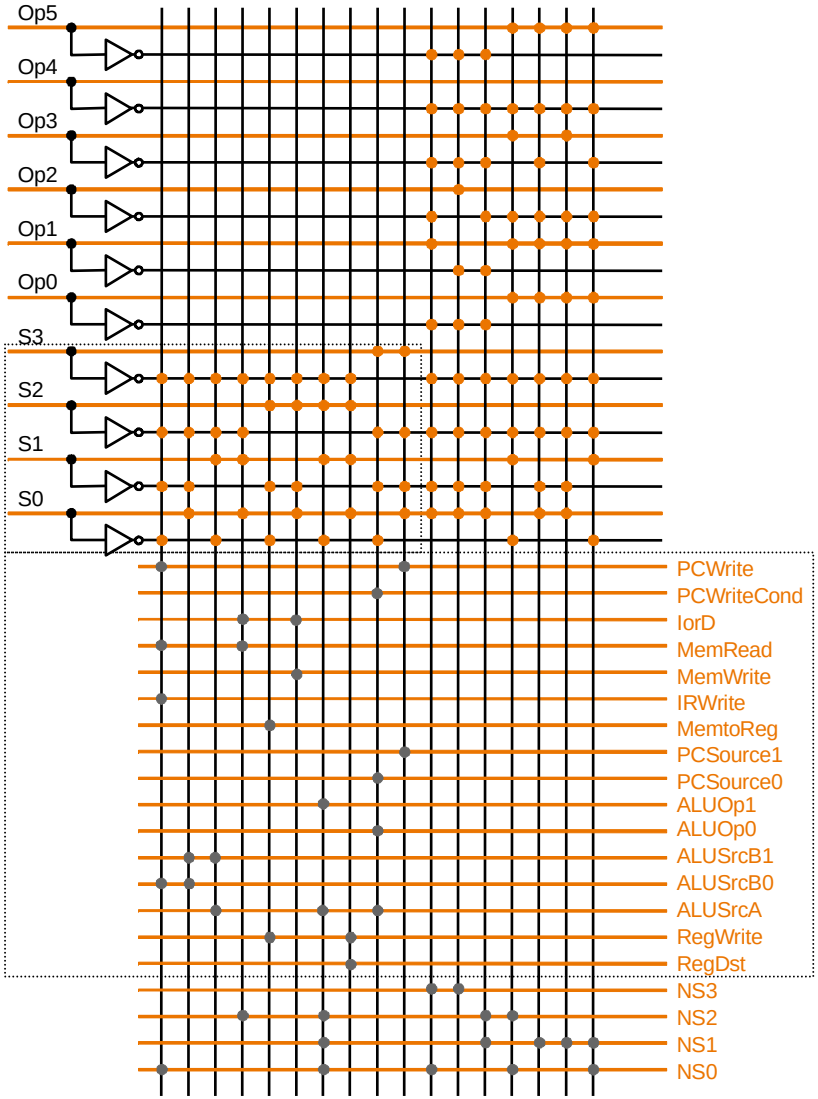
# Progetto dell'unità di controllo
# Macchina a stati finiti

**Implementazione**



# Progetto dell'unità di controllo
# Macchina a stati finiti

**Implementazione tramite PLA**



# Progetto dell'unità di controllo
# Macchina a stati finiti

**Implementazione tramite ROM**

È più dispendiosa, in quanto l'implementazione di esempio con 10 linee di ingresso e 20 di uscita richiede una ROM di $2^{10}$ x 20 = 20Kbit.

Una PLA richiede (ingressi x mintermini) + (uscite x mintermini) (10x17)+(20x17) = 510 celle.

Sebbene le celle di una PLA siano leggermente più grandi che in una ROM, la PLA è molto più piccola e quindi più veloce.

Trattandosi di un automa di Moore la funzione di uscita e la funzione di stato prossimo possono essere implementate separatamente consentendo così di dividere in due la PLA e la ROM, in tal caso si hanno rispettivamente 298 celle e 4,3 Kbit con ovvi benefici.

# Progetto dell’unità di controllo
# Microprogrammazione

La specifica grafica dell’unità di controllo diventa ingestibile al crescere del numero delle istruzioni e dei cicli di clock che ogni istruzione può richiedere.

Si introduce quindi una rappresentazione testuale che ha delle analogie con la programmazione.

Insiemi disgiunti di segnali di controllo sono rappresentati simbolicamente.

L’insieme dei segnali che devono essere affermati in un determinato stato viene chiamato **microistruzione**.

L’esecuzione di una microistruzione consiste quindi nell’affermare i segnali che sono specificati come campi del formato della microistruzione stessa.

Un campo apposito specifica come determinare la microistruzione successiva.

# Formato delle microistruzioni

| Field name | Value | Signals active | Comment |
|---|---|---|---|
| | Add | ALUOp = 00 | Cause the ALU to add. |
| ALU control | Subt | ALUOp = 01 | Cause the ALU to subtract; this implements the compare for |
| | | | branches. |
| | Func code | ALUOp = 10 | Use the instruction's function code to determine ALU control. |
| SRC1 | PC | ALUSrcA = 0 | Use the PC as the first ALU input. |
| | A | ALUSrcA = 1 | Register A is the first ALU input. |
| | B | ALUSrcB = 00 | Register B is the second ALU input. |
| SRC2 | 4 | ALUSrcB = 01 | Use 4 as the second ALU input. |
| | Extend | ALUSrcB = 10 | Use output of the sign extension unit as the second ALU input. |
| | Extshft | ALUSrcB = 11 | Use the output of the shift-by-two unit as the second ALU input. |
| | Read | | Read two registers using the rs and rt fields of the IR as the register |
| | | | numbers and putting the data into registers A and B. |
| | Write ALU | RegWrite, | Write a register using the rd field of the IR as the register number and |
| Register | | RegDst = 1, | the contents of the ALUOut as the data. |
| control | | MemtoReg = 0 | |
| | Write MDR | RegWrite, | Write a register using the rt field of the IR as the register number and |
| | | RegDst = 0, | the contents of the MDR as the data. |
| | | MemtoReg = 1 | |
| | Read PC | MemRead, IRWrite | Read memory using the PC as address; write result into IR (and |
| | | IorD = 0 | the MDR). |
| Memory | Read ALU | MemRead, | Read memory using the ALUOut as address; write result into MDR. |
| | | IorD = 1 | |
| | Write ALU | MemWrite, | Write memory using the ALUOut as address, contents of B as the |
| | | IorD = 1 | data. |
| | ALU | PCSource = 00, | Write the output of the ALU into the PC. |
| | | PCWrite | |
| PC write control | ALUOut-cond | PCSource = 01, | If the Zero output of the ALU is active, write the PC with the contents |
| | | PCWriteCond | of the register ALUOut. |
| | jump address | PCSource = 10, | Write the PC with the jump address from the instruction. |
| | | PCWrite | |
| | Seq | AddrCtl = 11 | Choose the next microinstruction sequentially. |
| Sequencing | Fetch | AddrCtl = 00 | Go to the first microinstruction to begin a new instruction. |
| | Dispatch 1 | AddrCtl = 01 | Dispatch using the ROM 1. |
| | Dispatch 2 | AddrCtl = 10 | Dispatch using the ROM 2. |

# Il microprogramma

| Label | ALU control | SRC1 | SRC2 | Register control | Memory | PCWrite control | Sequencing |
|---|---|---|---|---|---|---|---|
| Fetch | Add | PC | 4 | | Read PC | ALU | Seq |
| | Add | PC | Extshft | Read | | | Dispatch 1 |
| Mem1 | Add | A | Extend | | | | Dispatch 2 |
| LW2 | | | | | Read ALU | | Seq |
| | | | | Write MDR | | | Fetch |
| SW2 | | | | | Write ALU | | Fetch |
| Rformat1 | Func code | A | B | | | | Seq |
| | | | | Write ALU | | | Fetch |
| BEQ1 | Subt | A | B | | | ALUOut-cond | Fetch |
| JUMP1 | | | | | | Jump address | Fetch |

| Dispatch ROM 1 | | |
|---|---|---|
| Op | Opcode name | Value |
| 000000 | R-format | 0110 |
| 000010 | jmp | 1001 |
| 000100 | beq | 1000 |
| 100011 | lw | 0010 |
| 101011 | sw | 0010 |

| Dispatch ROM 2 | | |
|---|---|---|
| Op | Opcode name | Value |
| 100011 | lw | 0011 |
| 101011 | sw | 0101 |

# Il microprogramma

| | Label | ALU control | SRC1 | SRC2 | Register control | Memory | PCWrite control | Sequencing |
|---|---|---|---|---|---|---|---|---|
| 0 | Fetch | Add | PC | 4 | | Read PC | ALU | Seq |
| 1 | | Add | PC | Extshft | Read | | | Dispatch 1 |
| 2 | Mem1 | Add | A | Extend | | | | Dispatch 2 |
| 3 | LW2 | | | | | Read ALU | | Seq |
| 4 | | | | | Write MDR | | | Fetch |
| 5 | SW2 | | | | | Write ALU | | Fetch |
| 6 | Rformat1 | Func code | A | B | | | | Seq |
| 7 | | | | | Write ALU | | | Fetch |
| 8 | BEQ1 | Subt | A | B | | | ALUOut-cond | Fetch |
| 9 | JUMP1 | | | | | | Jump address | Fetch |

**0 fetch, 1**

**1 instruction-decode-register-fecth, 2 o 6 o 8 o 9**

**2 memory-address-computation, 3 o 5**

**3 memory-access-read, 4**

**4 memory-read-completion, 0**

**5 memory-access-write, 0**

**6 execution, 7**

**7 r-type-completion, 0**

**8 branch-completion, 0**

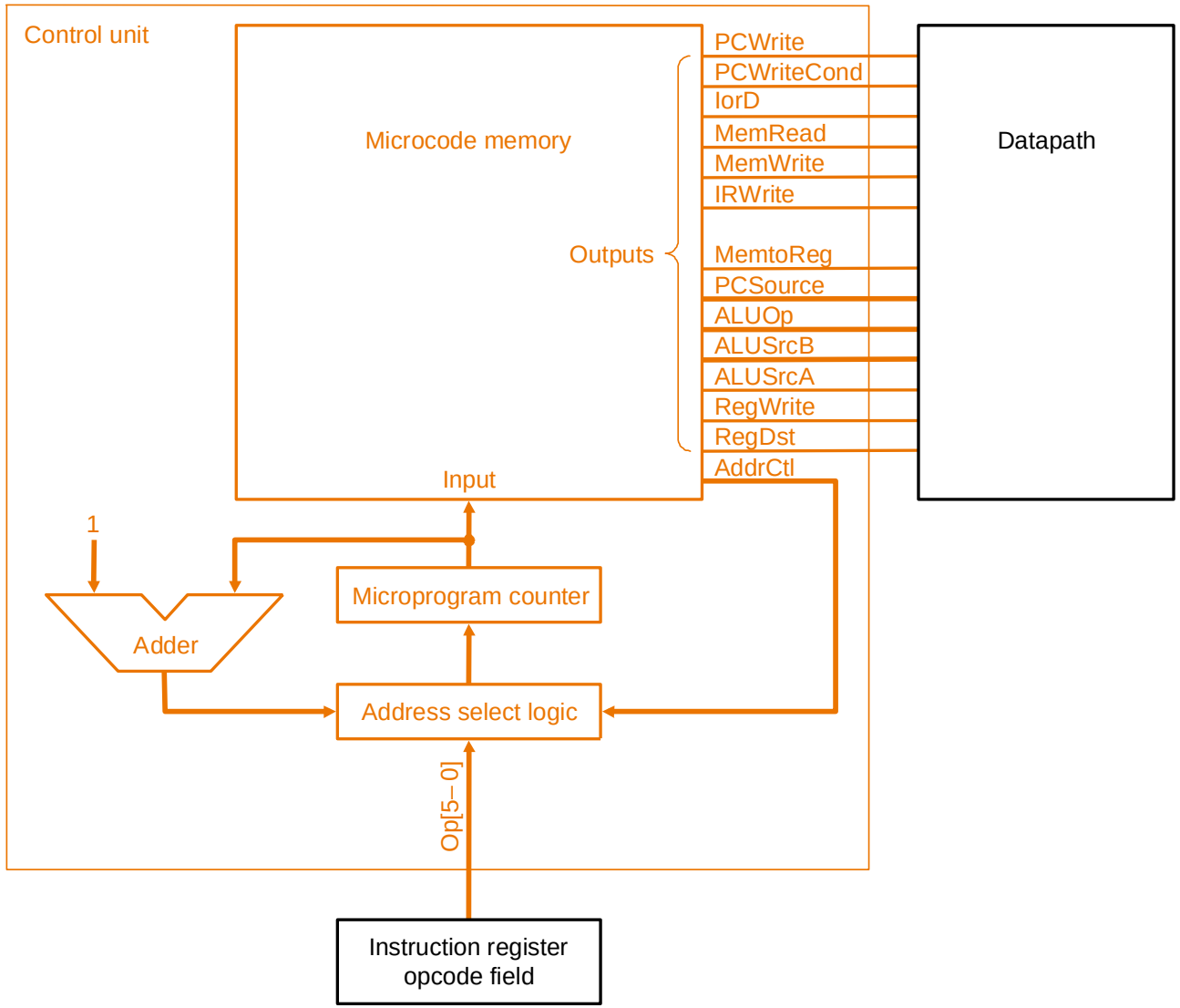
**9 jump-completion, 0**

# Il microprogramma

| Label | ALU control | SRC1 | SRC2 | Register control | Memory | PCWrite control | Sequencing |
|---|---|---|---|---|---|---|---|
| Fetch | Add | PC | 4 | | Read PC | ALU | Seq |
| | Add | PC | Extshft | Read | | | Dispatch 1 |
| Mem1 | Add | A | Extend | | | | Dispatch 2 |
| LW2 | | | | | Read ALU | | Seq |
| | | | | Write MDR | | | Fetch |
| SW2 | | | | | Write ALU | | Fetch |
| Rformat1 | Func code | A | B | | | | Seq |
| | | | | Write ALU | | | Fetch |
| BEQ1 | Subt | A | B | | | ALUOut-cond | Fetch |
| JUMP1 | | | | | | Jump address | Fetch |

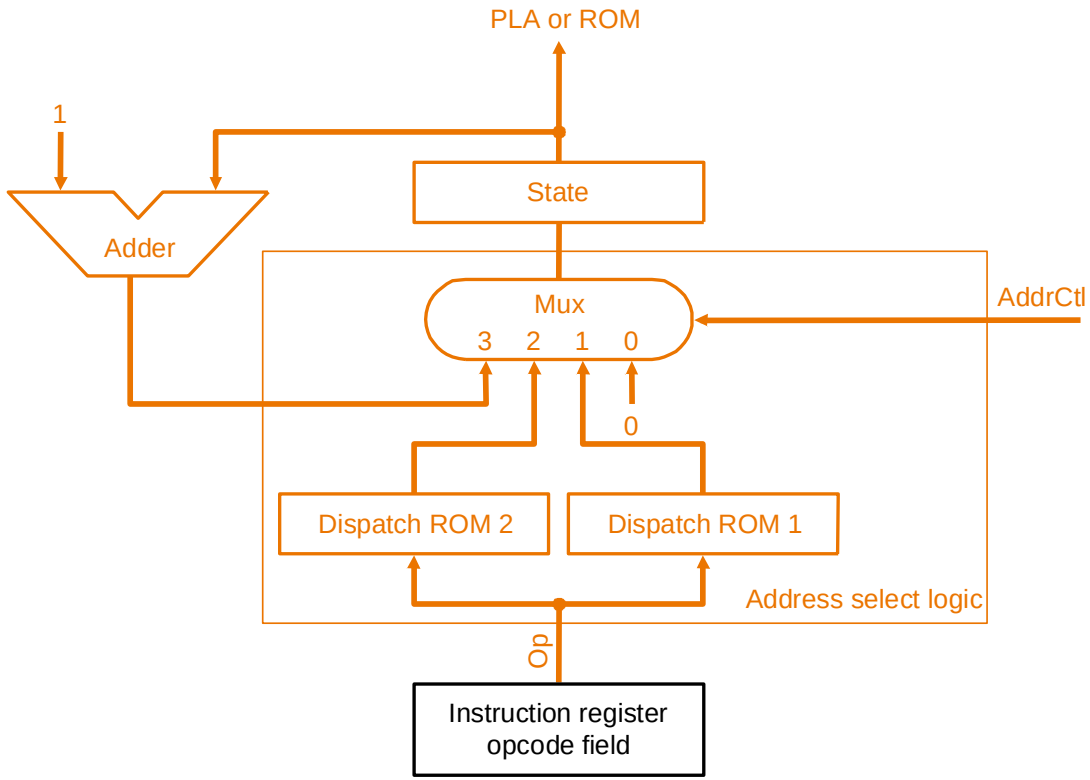
| State number | Address-control action | Value of AddrCtl |
|---|---|---|
| 0 | Use incremented state | 3 |
| 1 | Use dispatch ROM 1 | 1 |
| 2 | Use dispatch ROM 2 | 2 |
| 3 | Use incremented state | 3 |
| 4 | Replace state number by 0 | 0 |
| 5 | Replace state number by 0 | 0 |
| 6 | Use incremented state | 3 |
| 7 | Replace state number by 0 | 0 |
| 8 | Replace state number by 0 | 0 |
| 9 | Replace state number by 0 | 0 |

# Implementazione del microprogramma



# Implementazione del microprogramma

## Logica di selezione dell'indirizzo



# Vantaggi e svantaggi della microprogrammazione

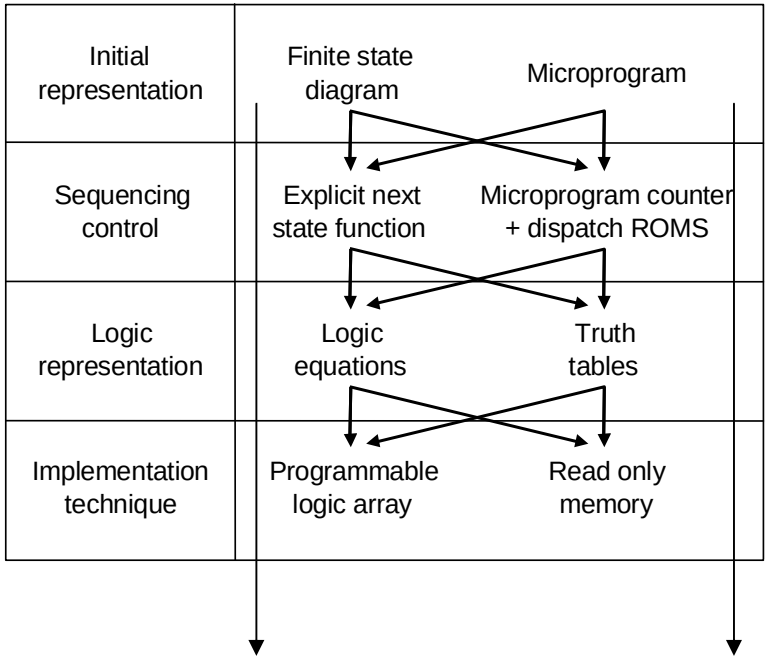
*La distinzione tra specifica ed implementazione è qualche volta sfumata…*

- Vantaggi nella specifica:
  - Facile da progettare e scrivere.
  - Progettazione dell'architettura e del codice in parallelo.
- Vantaggi dell'implementazione (con ROM separata)
  - Facile da modificare.
  - Possibilità di simulare altre architetture.
- Svantaggi nell'implementazione ora che…
  - Il controllo è implementato nello stesso chip come processore.
  - Le ROM non sono più veloci delle RAM.
  - Non c'è più necessità di correzione grazie a strumenti CAD.

# Livelli di astrazione

**Utente**

# Metodi per la specifica e l'implementazione dell'unità di controllo




**Hardwired control** **Microprogrammed control**

Unità di controllo cablata: storicamente ricevette questo nome perché la funzione di controllo era implementata in hardware e non era facilmente modificabile in contrapposizione al microcodice memorizzato in una ROM (detto anche *firmware*).

# Unità di controllo nell'architettura 80x86

L'architettura Intel 80x86 contiene istruzioni molto complesse (CISC), che possono richiedere fino a centinaia di cicli di clock.

Un'unità di controllo microprogrammata semplifica l'implementazione a scapito delle prestazioni.

A partire dal 486 è stata utilizzata un'unità di controllo cablata per le istruzioni più semplici che richiedono pochi cicli di clock ed un'unità di controllo microprogrammata per le istruzioni più complesse.

# Eccezioni

Eventi inattesi che cambiano il normale flusso di esecuzione delle istruzioni.

**Eccezione**, se proviene dall'interno del processore.

**Interruzione** (**Interrupt**), se proviene dall'esterno del processore.

Significati diversi a seconda dell'architettura…

MIPS: eccezione per tutti gli eventi ed interrupt per gli eventi esterni.

Intel 80x86: interrupt per tutti gli eventi.

# Gestione delle eccezioni

In riferimento all'implementazione vista possono verificarsi due tipi di eccezioni: istruzione non valida ed overflow.

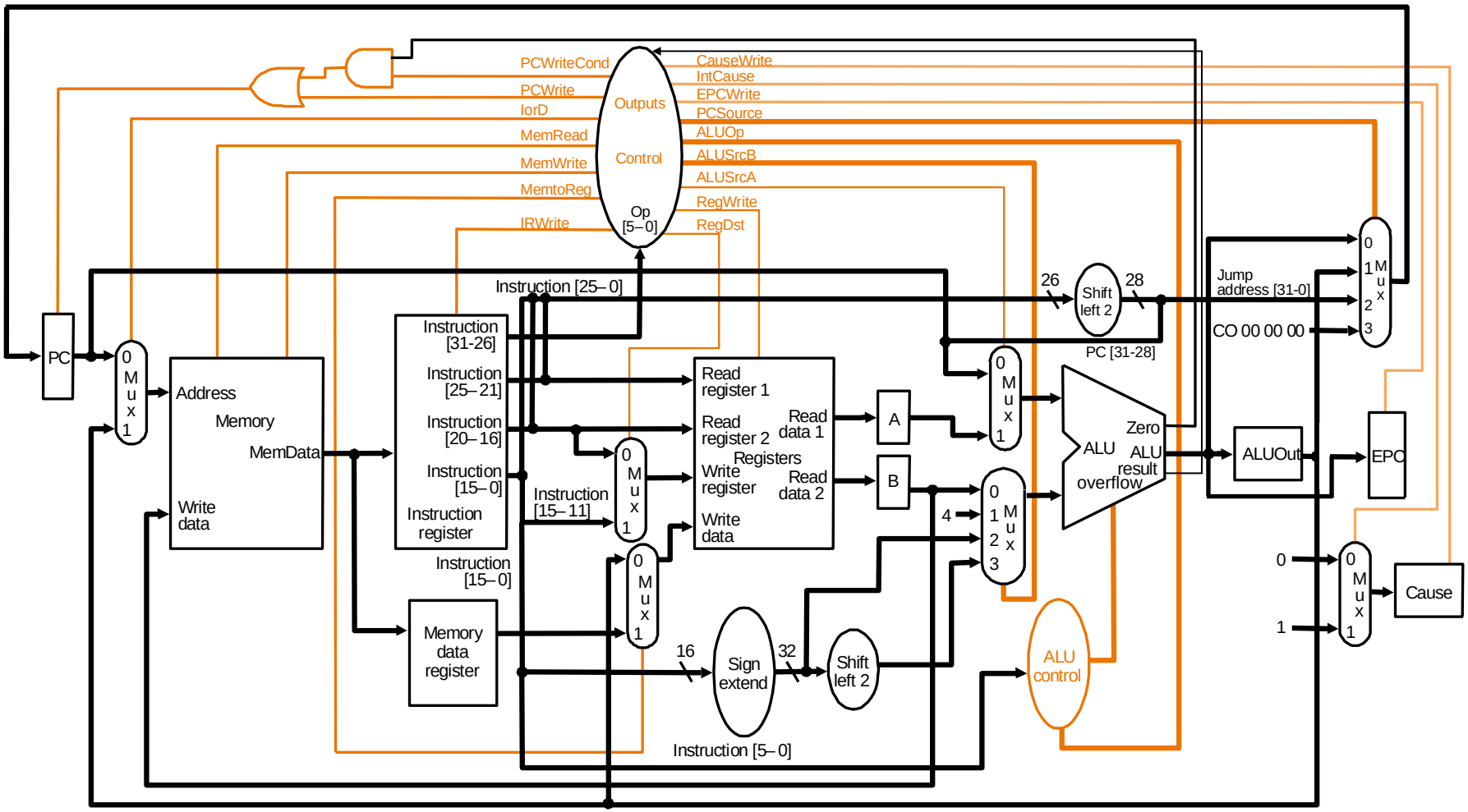
Il processore deve salvare l'indirizzo dell'istruzione, in cui si è verificata l'eccezione, nell'**Exception Program Counter** (**EPC**) e trasferire il controllo al sistema operativo.

## Causa dell'eccezione

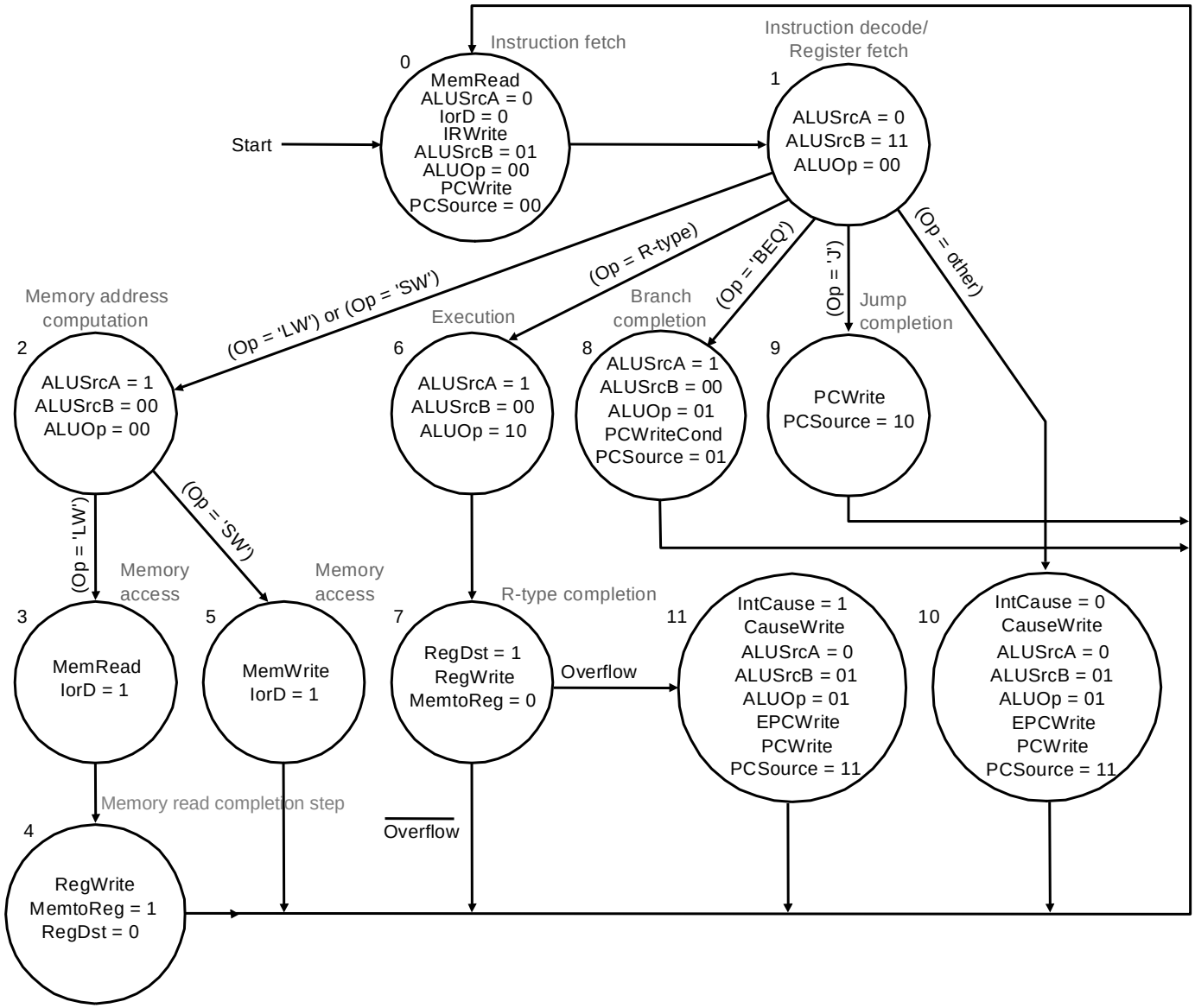
1. Registro causa (utilizzato nell'architettura MIPS)
2. Interruzioni vettorizzate – vectored interrupts (la causa dell'eccezione determina l'indirizzo a cui trasferire il controllo).

Estendiamo l'unità di elaborazione e di controllo per gestire le suddette due eccezioni utilizzando il bit meno significativo del registro causa (0: istruzione non definita, 1:overflow)

# Gestione delle eccezioni



# Gestione delle eccezioni



# Confronto di prestazioni fra diverse versioni di CPU multi-ciclo

- Si ipotizzi di avere un programma con la seguente combinazione di istruzioni:
  - 24% di load, 12% di store, 44% di istruzioni formato-R, 18% di salti condizionati e 2% di salti incondizionati
- Si vogliono confrontare 3 CPU multi-ciclo aventi le seguenti caratteristiche:

  **M1**: stessa unità di elaborazione multi-ciclo vista finora con frequenza di clock 500 MHz

  **M2**: unità simile a quella di M1, ma dove gli aggiornamenti dei registri vengono fatti nello stesso ciclo di clock di una lettura in memoria o di un'operazione con la ALU. Frequenza di clock pari a 400 MHz.

  **M3**: unità simile a quella di M2, ma dove i calcoli degli indirizzi vengono fatti nello stesso ciclo di clock dell'accesso alla memoria. Frequenza di clock pari a 250 MHz

  (La frequenza di clock nel caso di M2 ed M3 diminuisce a causa dell'aumentare delle operazioni da svolgere nel singolo ciclo di clock)

# Tempo di CPU per M1

Numero di cicli per ciascuna classe di istruzioni

- Load = 5
- Store = 4
- Formato-R = 4
- Salti cond. = 3
- Salti incond. = 3

$$CPI = 0{,}24\times5 + 0{,}12\times4 + 0{,}44\times4 + 0{,}18\times3 + 0{,}02\times3 = 4{,}04$$

$$\text{Tempo di } CPU_{M1} = \frac{I\times CPI}{\text{frequenza di clock}} = \frac{I\times 4{,}04\,\text{cicli}}{500\times10^{6}\,\text{cicli}/\text{sec}}$$

# Tempo di CPU per M2

- **M2**: unità simile a quella di M1, ma dove gli aggiornamenti dei registri vengono fatti nello stesso ciclo di clock di una lettura in memoria o di un'operazione con la ALU.
- Questo significa che i passi 3 e 4 (accesso in memoria e scrittura nel registro) dell'automa a stati finiti (fig. 5.42) vengono fusi, così come i passi 6 e 7 (operazione ALU e scrittura del risultato in un registro).
- Quindi, il numero di cicli per ciascuna classe di istruzioni diventa:
  Load = 4
  Store = 4
  Formato-R = 3
  Salti cond. = 3
  Salti incond. = 3
- Da cui **CPI** = 0,24×4 + 0,12×4 + 0,44×3 + 0,18×3 + 0,02×3 = 3,36

$$\text{Tempo di CPU}_{M2} = \frac{I \times CPI}{\text{frequenza di clock}} = \frac{I \times 3{,}36\ \text{cicli}}{400 \times 10^6\ \text{cicli/sec}}$$

# Tempo di CPU per M3

- **M3**: unità simile a quella di M2, ma dove i calcoli degli indirizzi vengono fatti nello stesso ciclo di clock dell'accesso alla memoria.
- Questo significa che i passi 3 e 4 vengono fusi anche col passo 2 (calcolo degli indirizzi), il passo 5 viene fuso anch'esso col passo 2, e i passi 6 e 7 vengono fusi come in M2.
- Quindi, il numero di cicli per ciascuna classe di istruzioni diventa:
  Load = 3
  Store = 3
  Formato-R = 3
  Salti cond. = 3
  Salti incond. = 3
- Da cui **CPI** = 0,24×3 + 0,12×3 + 0,44×3 + 0,18×3 + 0,02×3 = 3

$$\text{Tempo di CPU}_{M3} = \frac{I \times CPI}{\text{frequenza di clock}} = \frac{I \times 3\,\text{cicli}}{250 \times 10^6\,\text{cicli/sec}}$$

# Confronto fra le 3 CPU multi-ciclo

$$\frac{\text{Tempo di CPU}_{M2}}{\text{Tempo di CPU}_{M1}} = \frac{3{,}36}{400\times10^6} \times \frac{500\times10^6}{4{,}04} = 1{,}04$$

**M1 è più veloce di M2 di 1,04 volte**

$$\frac{\text{Tempo di CPU}_{M3}}{\text{Tempo di CPU}_{M1}} = \frac{3}{250\times10^6} \times \frac{500\times10^6}{4{,}04} = 1{,}49$$

**M1 è più veloce di M3 di 1,49 volte**

$$\frac{\text{Tempo di CPU}_{M3}}{\text{Tempo di CPU}_{M2}} = \frac{3}{250\times10^6} \times \frac{400\times10^6}{3{,}36} = 1{,}43$$

**M2 è più veloce di M3 di 1,43 volte**

# Riferimenti

**Computer Organization and Design**

**The Hardware/Software Interface 3rd Edition**

David A. Patterson, John L. Hennessy

*Capitolo 5*

Versione italiana:

**Struttura e Progetto dei Calcolatori**

**L'Interfaccia Hardware-Software**

**2ª edizione Zanichelli**

http://en.wikipedia.org/ o http://it.wikipedia.org/